CPSL
Cyber-Physical Systems Laboratory

Progressive Neural Compression for Adaptive Image Offloading under Timing Constraints

Ruiqi Wang, Hanyang Liu, Jiaming Qiu, Moran Xu, Roch Guérin, Chenyang Lu

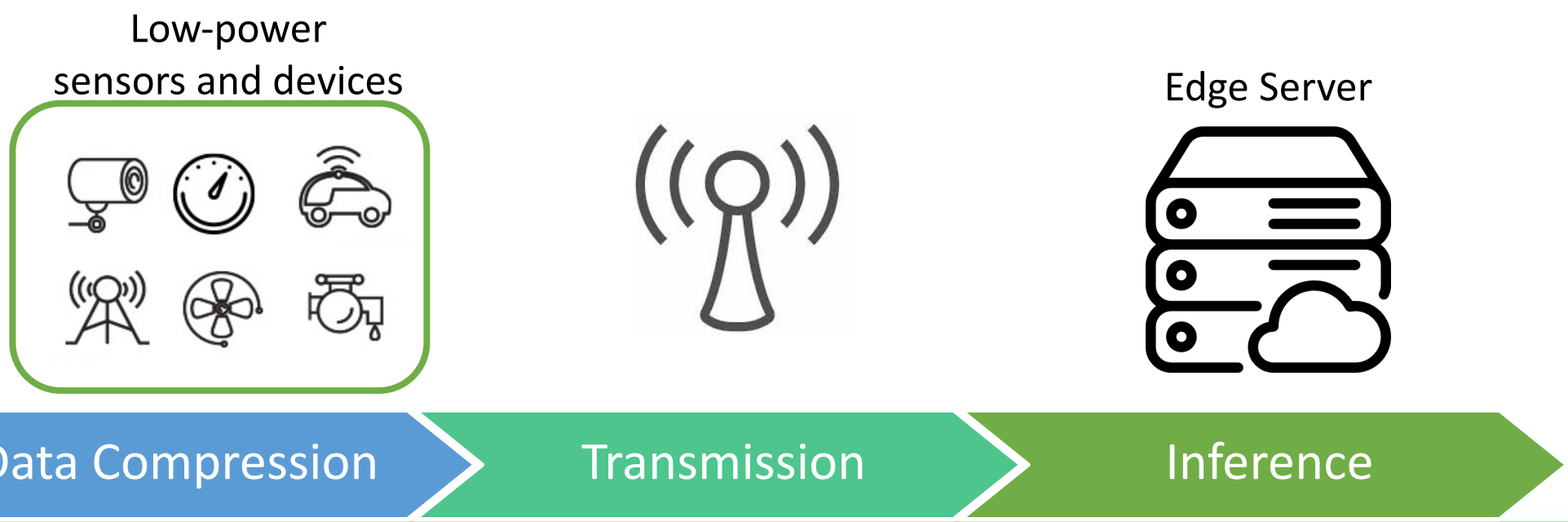
Department of Computer Science and Engineering

Washington University in St. Louis

Washington University in St. Louis

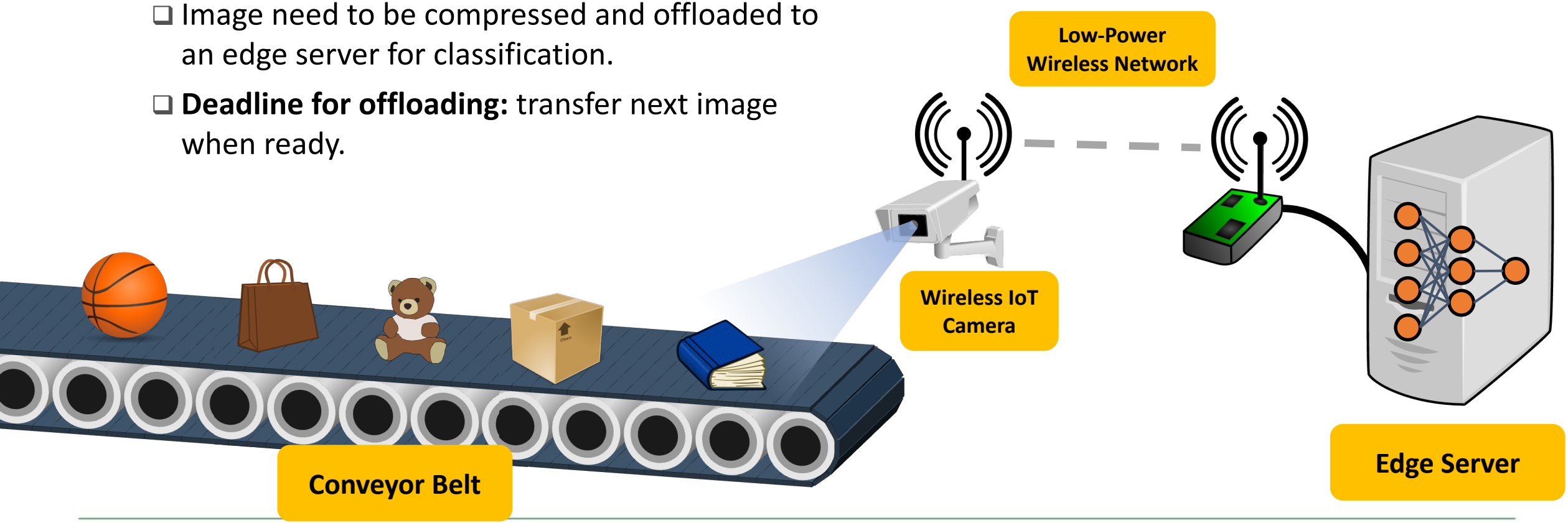
Background

- In real-time AI systems, inference tasks often need to be completed by a certain deadline.
  - e.g., the arrival of the next image or to allow actuation to be performed in time.
- In an edge-computing setting, sensors collect, compress and offload the data to an edge server for classification.



A Motivating Application

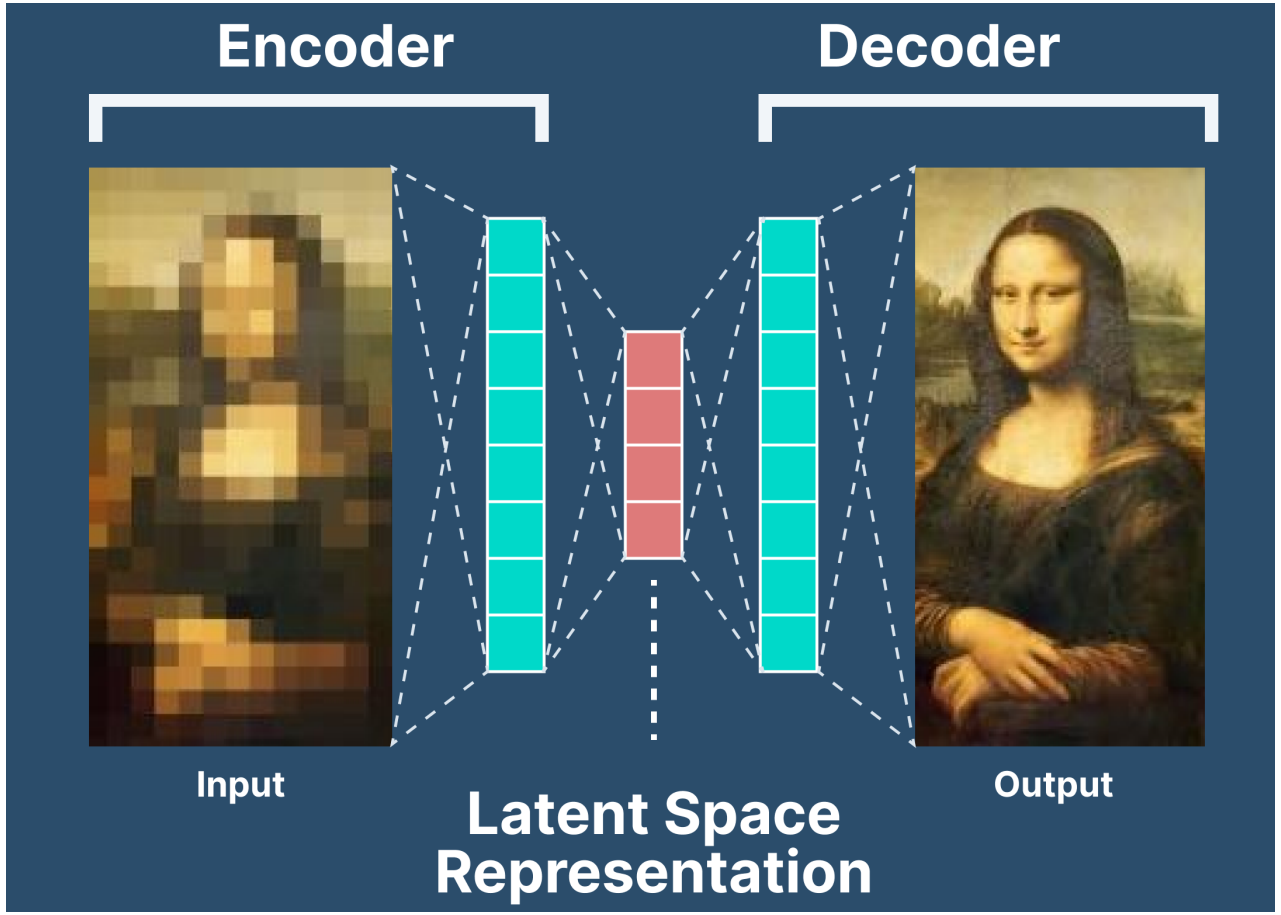
- In our work, we take image classification as a target application.
- For example, a conveyor belt in a warehouse:
  - An IoT camera capture images of each object.
  - Image need to be compressed and offloaded to an edge server for classification.
  - **Deadline for offloading:** transfer next image when ready.



Problem Formulation

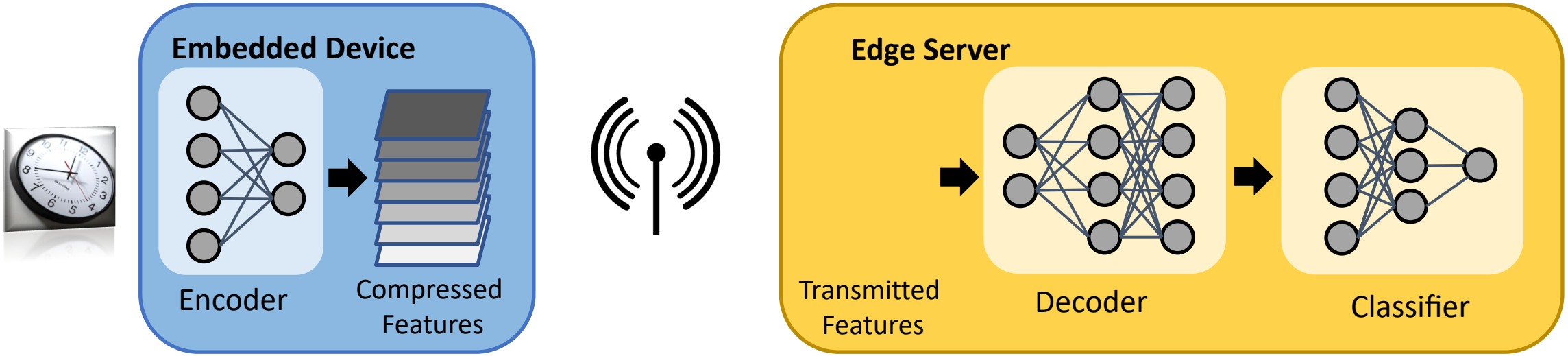
Autoencoder (AE)

- An autoencoder (AE) is a type of artificial neural network used to learn data encodings in an unsupervised manner.
- AE compresses data into features with an encoder network into **latent space representation (data of smaller size)** and reconstructs the data with a decoder.
- An efficient method to compress data.



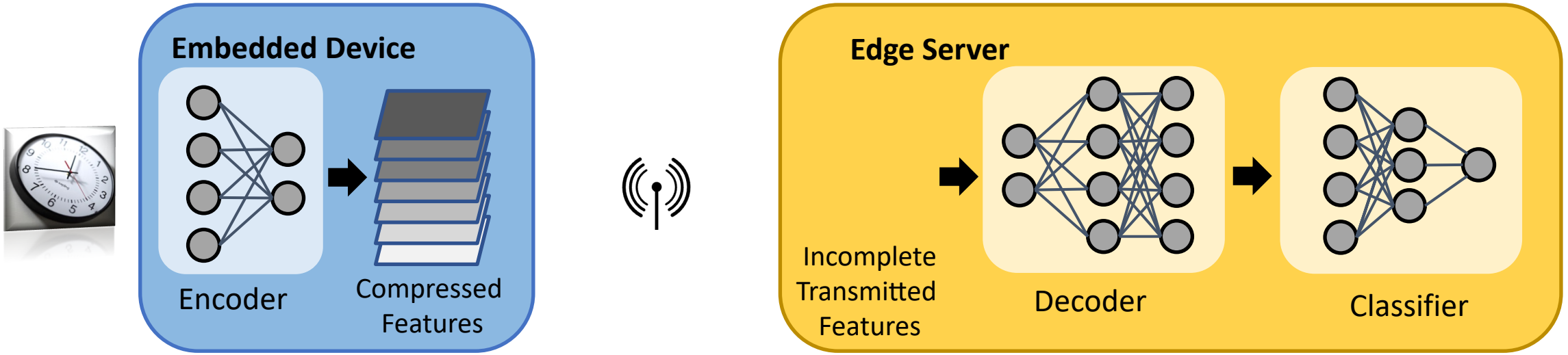
AE in an Edge Computing Setting

- In an edge computing setting with desirable offloading condition.



When the wireless network bandwidth is **high**, the embedded device can offload all the compressed features.

AE in an Edge Computing Setting

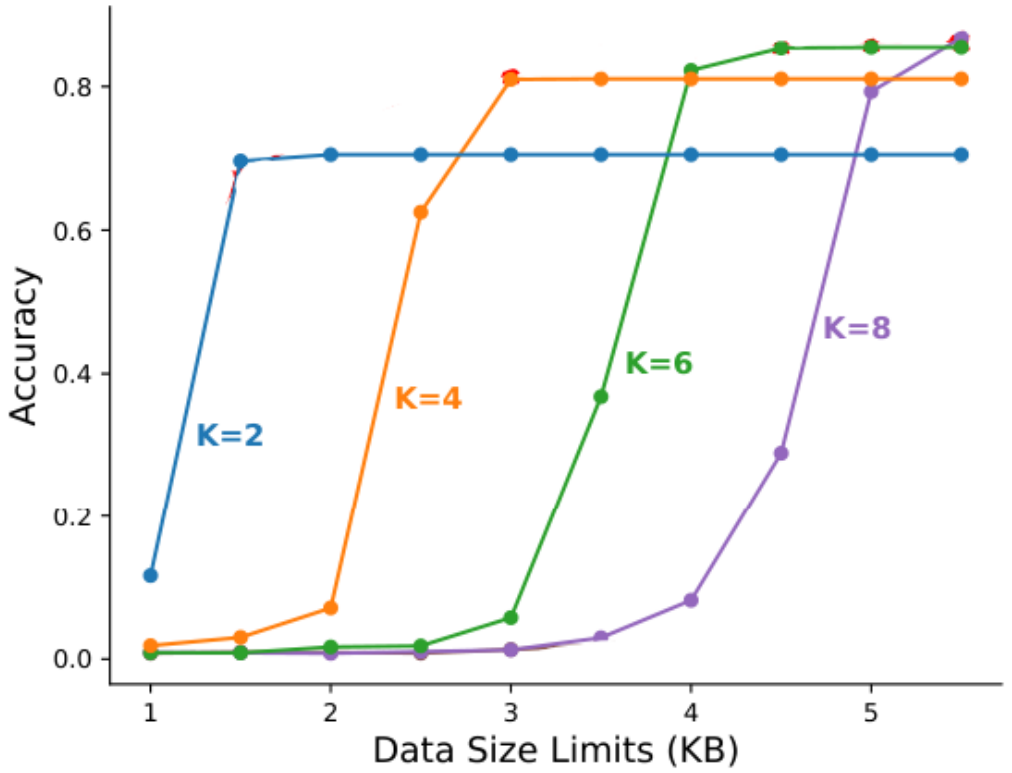


When the wireless network bandwidth is **low**, the embedded device may only offload some of the features within the deadline.

AE in an Edge Computing Setting

- Any problem with regular AE when the data received is incomplete?

- When data are incomplete.
- Abrupt accuracy drop!



Contribution of this paper

	Traditional Image Encoding	Neural Compression
Non-progressive	JPEG, WebP, etc.	Autoencoder-based methods: DeepCOD (Yao et al., 2020), Starfish (Hu et al., 2020)
Progressive	Progressive JPEG	RNN-based (Toderici et al., 2017) **PNC (Our work)**

Contribution: We proposed Progressive Neural Compression (PNC) for IoT devices that need to offload images to edge servers for classification within a deadline despite varying network bandwidth.

- ***Progressive***, classification can be performed at any time based only on the data received up to that point, with the receipt of more data translating into progressively more accurate classification decisions.

Problem Formulation

Given:







- A periodic image sequence
- A deadline for transmission.
- Fluctuating bandwidth



Image $\mathbf{X}_i$

Embedded Device

Encoder f

Compressed Features

$$\mathbf{z}_i = f(\mathbf{x}_i)$$

S_i

Edge Server

Partially Transmitted Features

Decoder g

Classifier h

$$\tilde{\mathbf{z}}_i = \texttt{truncate}\,(\mathbf{z}_i, S_i)$$

Classification Result

$$a_i = h\,(g(\tilde{\mathbf{z}}_i))$$

- **Goal:** Maximize the image classification accuracy.

Motivation of Progressive Neural Compression

- An "open-loop" solution that optimize classification performance given the amount of data transmissions the network allows by the deadline.

- We identify the following requirements for an image offloading framework:
  - *Classification* accuracy
    - ✖ Image Reconstruction Quality.
    - ✓ Top-K accuracy.
  - Encoding efficiency
    - Minimal computation overhead on IoT devices.
  - Adaptation to different deadlines and varying bandwidth
    - Classification accuracy should improve as more data are received by the deadline

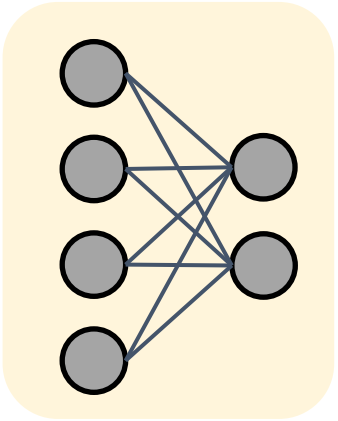
CPSL
Cyber-Physical Systems Laboratory

PNC

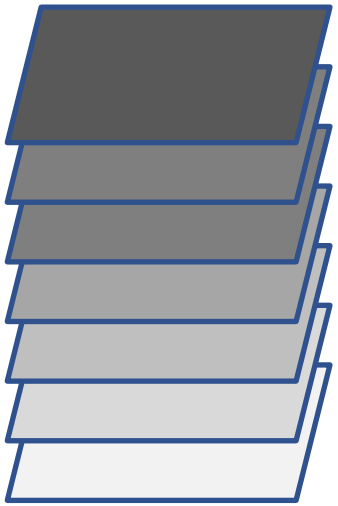
An illustrative example

- How can we deal with the loss of data?





Encoder



Compressed Features

Offloaded more often

Most Important

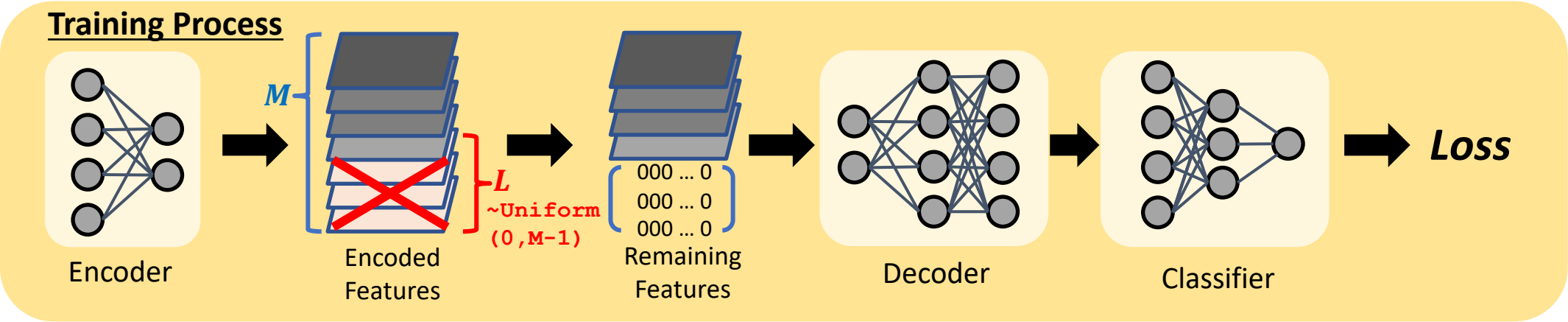
Offloaded less often

Least Important

We want to train an autoencoder so that features “on the top” have larger contributions to classification accuracy (higher importance) than the features “at the bottom.”

Tail-drop

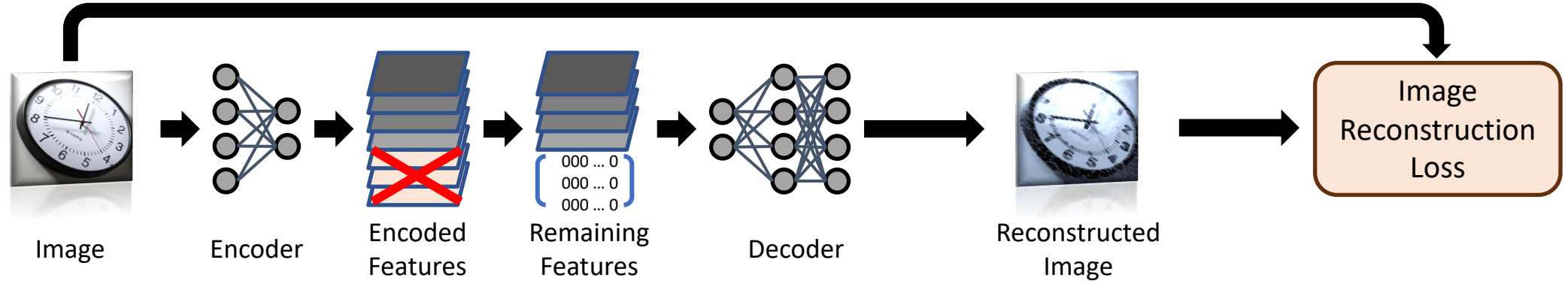
- "Stochastic Tail-drop" (Koike-Akino et al., 2020)
  - During each training iteration: **Randomly** zero-out the last L **features (out of** M**)**.
  - Optimize the training loss as normal.



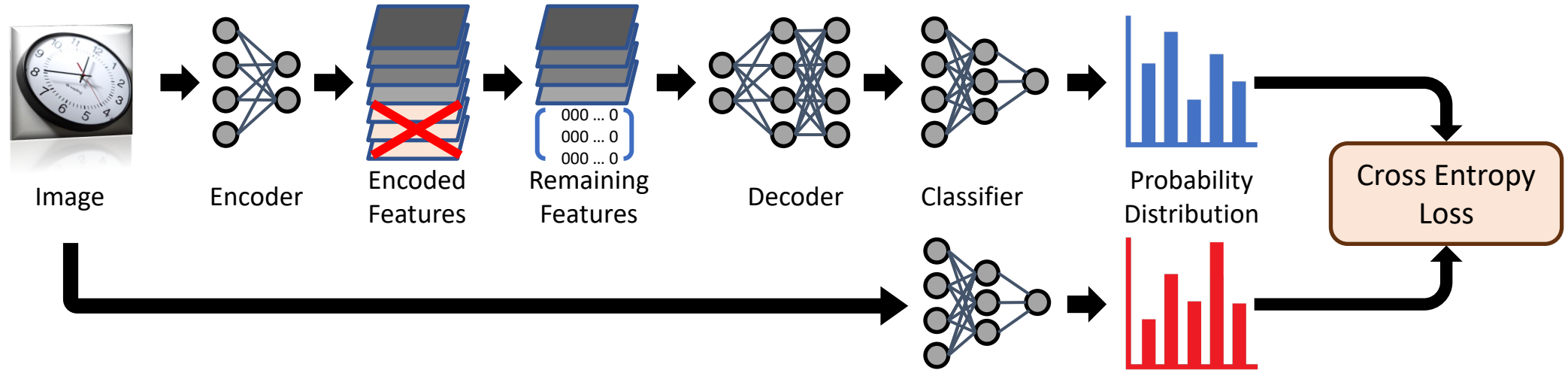
- By applying various Tail-drop length in different iterations:
  - Top features are trained more often, i.e., higher importance.
  - The decoder is also trained to deal with incomplete set of features.

Two stage training

- Stage 1: Unsupervised Pretraining

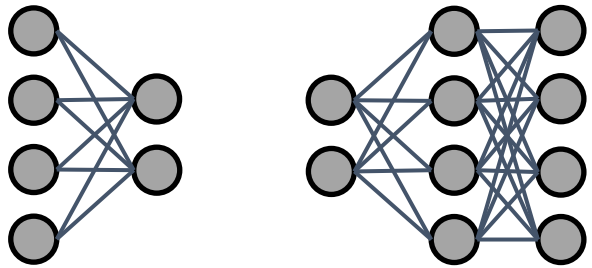


- Stage 2: Knowledge Distillation for Inference



Quantization and Huffman Coding

- Now, we have a pair of AE leading to reasonable downstream classification accuracy.



- Anything missing?
  - Further quantization (as we mentioned in the CV tutorial).
  - Our target device is an embedded system.
  - Models are trained on Tensorflow with 32-bit floating point precision. (Slow on RPI.)

Quantization

- **Model Quantization:** We use the automated TensorFlow Lite (TFLite) converter for 8-bit integer quantization.
  - Speeds up the encoding on resource-constrained devices
  - Reduce the encoded data size. (fp32 → uint8, 75% data reduced.)

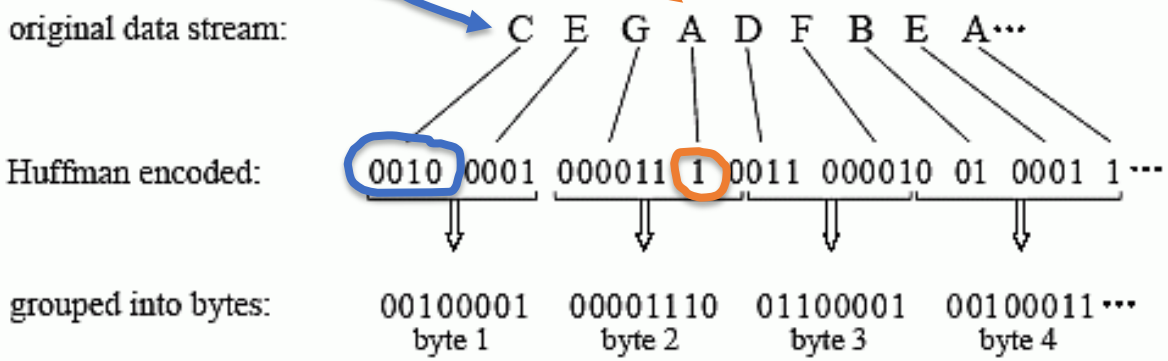
- **Feature Quantization:** We also realize that 8-bit is not the limit!
  - 8-bit integers (0-255) → 64 values represented by 6 bits
  - ~1% accuracy cost.
  - From 8-bit to 6-bit (25% data reduced.)

Huffman coding

- Huffman Coding:
  - one type of lossless data compression. 😄
  - **Application:** various compression formats like ZIP, JPEG, and MP3.
- Basic Concept:
  - More frequent symbols are assigned shorter codes.
  - Less frequent symbols are assigned longer codes.

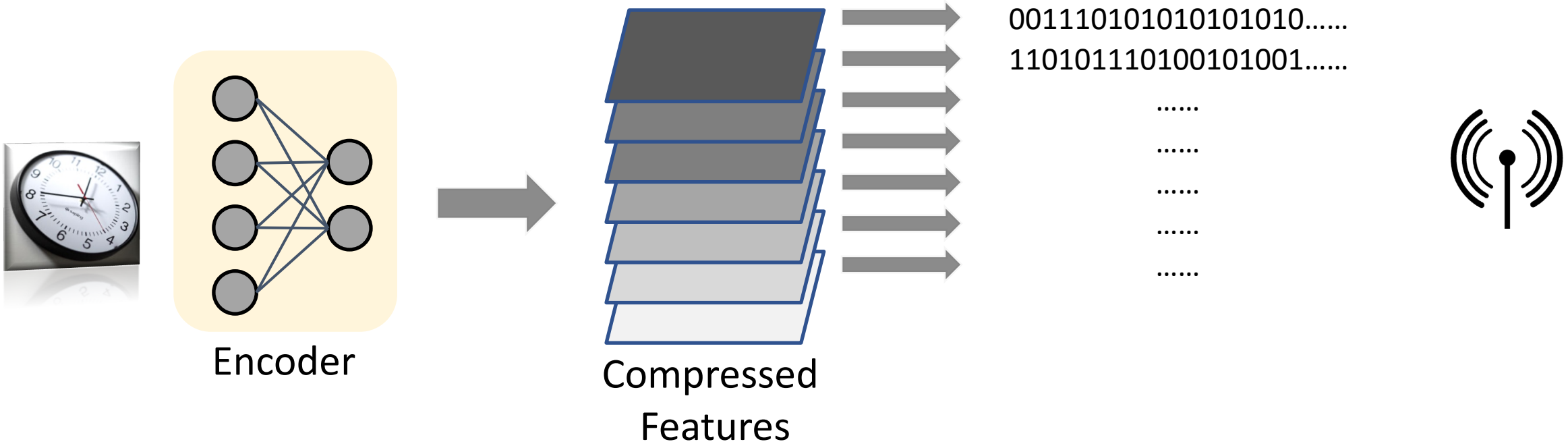
Example Encoding Table

letter	probability	Huffman code
A	.154	1
B	.110	01
C	.072	0010
D	.063	0011
E	.059	0001
F	.015	000010
G	.011	000011



Huffman coding

- As a result, we created one Huffman table for each channel of features.

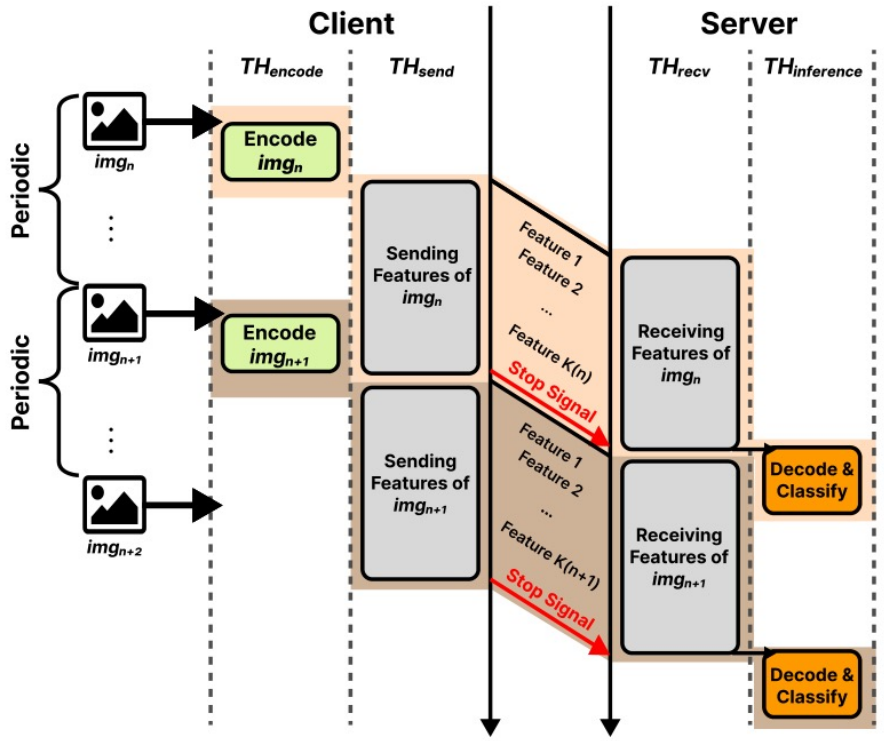


- The feature will be ready to transmit after Huffman coding.
- Then we can start the coding for the next feature.

Implementation

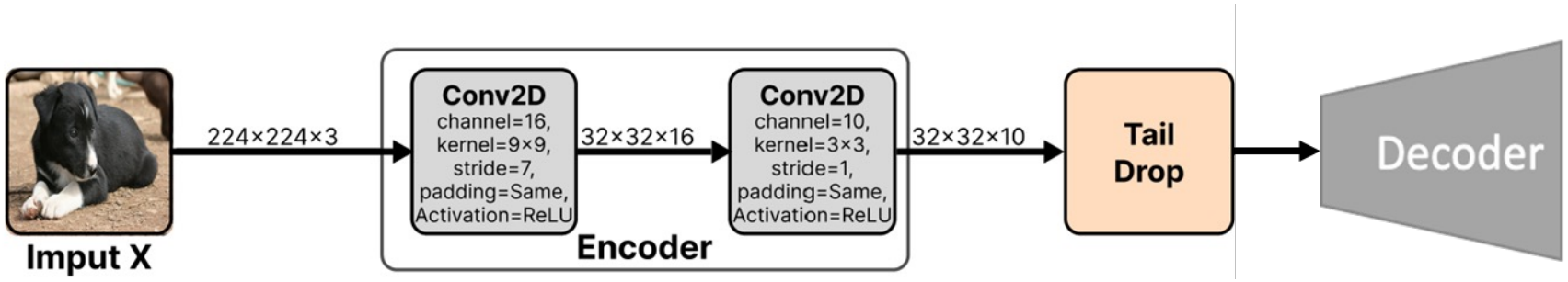
Progressive Image Offloading Pipeline

- A distributed system architecture comprising multiple stages:
  - Image encoding on the IoT device (client)
  - offloading
  - decoding and classification on the edge server
- Multi-threads to improve resource utilization.
- Features are sent in 64-byte data blocks until the deadline.
- As PNC terminates the offloading upon the first feature of the next image is encoded, all the timing constraints shall be met.



Model Implementation

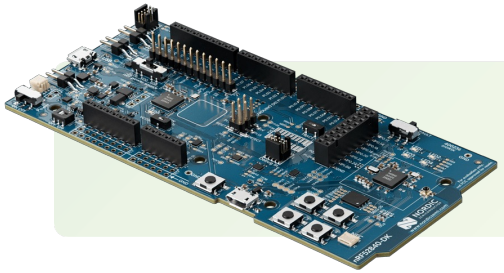
- We implement the PNC autoencoder with TensorFlow with an asymmetric design.
  - **Encoder: 2-layer convolutional network** to match the capacity of the embedded device.
  - **Decoder:** 5-layer convolutional network with higher dimension and complexity.
- Uniform distribution for tail length selection, as in the prior work.



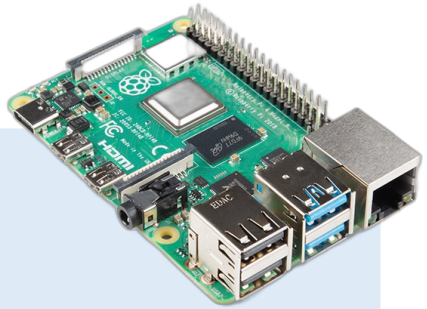
- Additional training data from the ImageNet validation set
  - 35,000/5,000 for training and validation
  - 2,000 for testbed experiments.

Testbed Implementation

- Hardware and Floor Plan for the experiments.

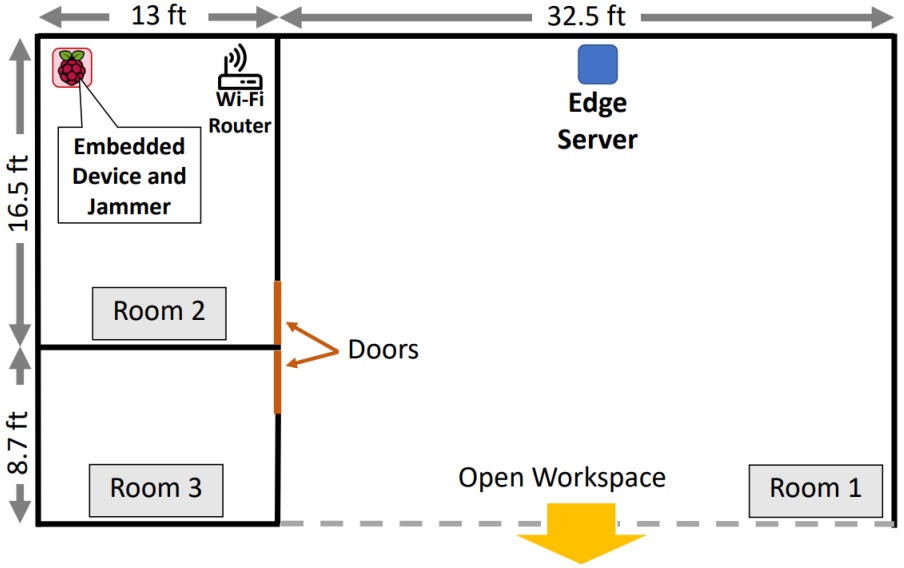


IEEE 802.15.4 radio module:
- A pair of nRF52840 development kits



Embedded IoT device:
- Raspberry Pi 4B

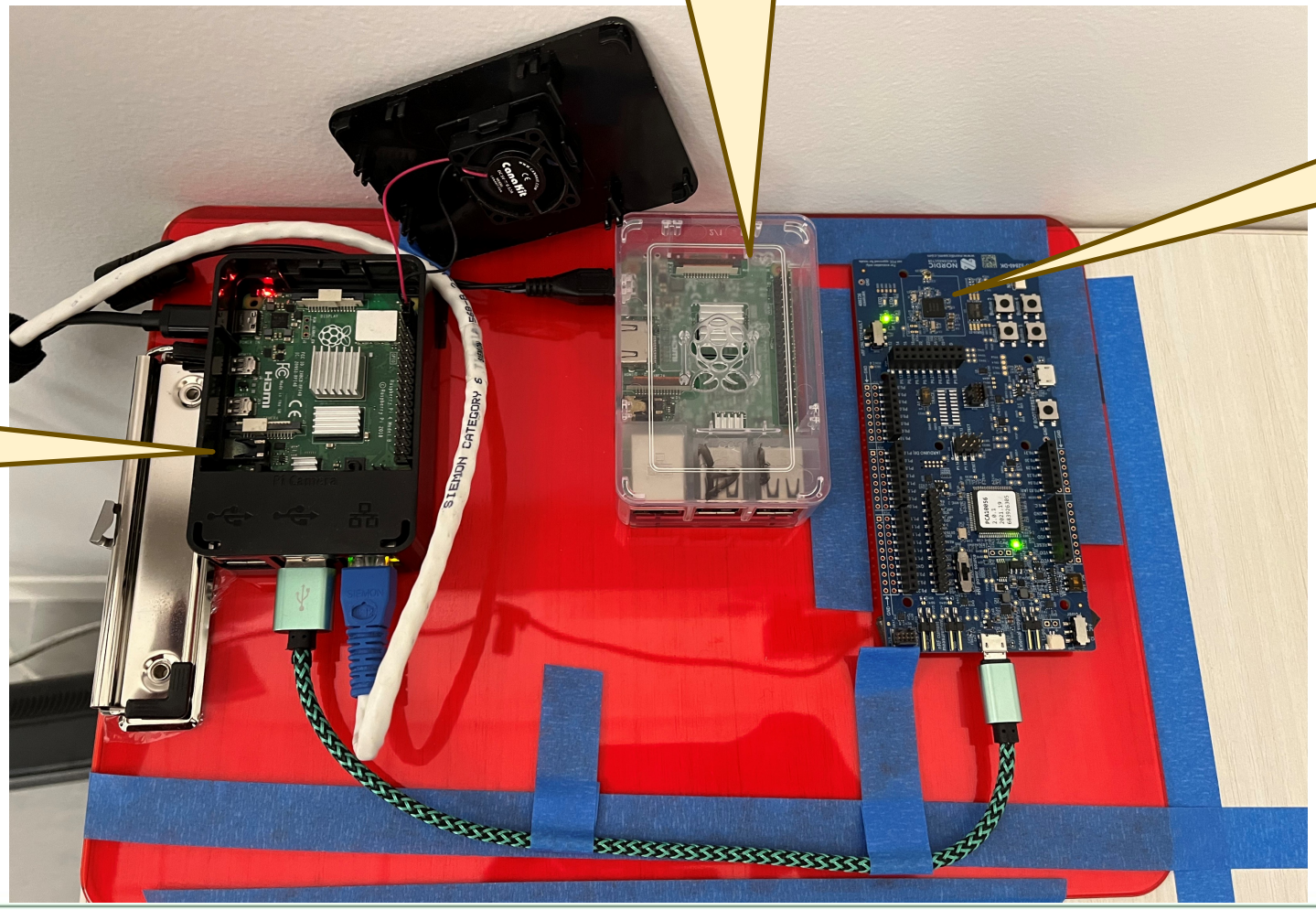
Jammer:
- Raspberry Pi 3B



- **Figure: Layout of the indoor space**

Edge Server:
- CPU: Intel Core i7-10700K
- GPU: Nvidia GeForce RTX 3090

Testbed Implementation



Experimental Setup

- **Network Interference:**
  - The device offloads images to the server over an IEEE 802.15.4 (2.4GHz) wireless link.
  - The Raspberry Pi "Jammer" generates 2.4GHz WiFi traffic at different rates using iperf3.
    - Overlapped frequency → network interference/jamming.
  - Three network interference scenarios:
    - **Scenario 1 (No Jamming):** There is no jamming traffic.
    - **Scenario 2 (Light Jamming):** `iperf3` on the jammer generates traffic at a rate of 10 Mbps.
    - **Scenario 3 (Heavy Jamming):** `iperf3` on the jammer generates traffic at a rate of 20 Mbps.
- **Image arrival period of** 300, 500, 700 ms.
- In total, 9 scenarios of various network condition and deadlines.

Evaluation

Baselines

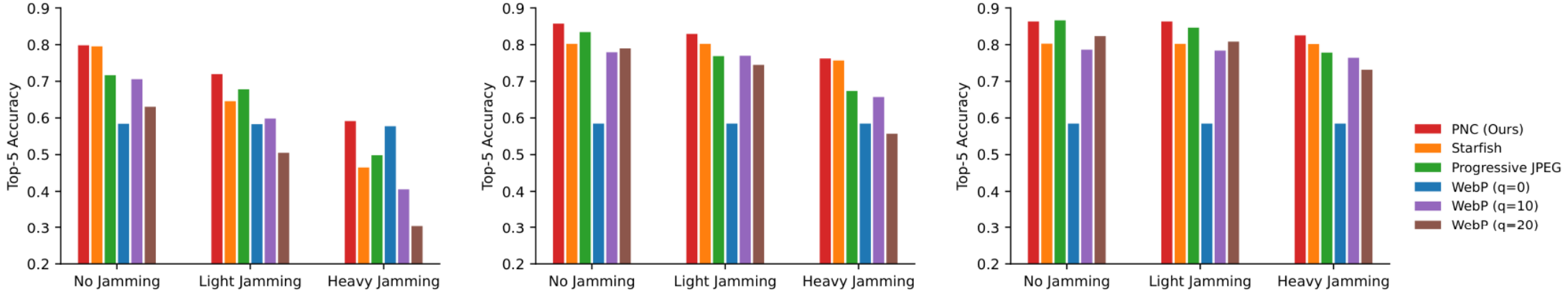
- **State of the Art and Baselines**
  - WebP
    - Efficient conventional image encoder, non-progressive.
  - Progressive JPEG
    - Progressive, conventional image encoder.
  - RNN-TFLite (Toderici et al., 2017)
    - Progressive, neural-network-based encoder.
  - Starfish (Hu et al., 2020)
    - Neural-network-based encoder with toleration for data loss.
- **Metric: Top-5 Accuracy**
  - Top-5 accuracy assigns a score (reward) of 1 for the received image if its ground-truth class is among the 5 most likely classes produced by the classifier and 0 otherwise.

Recall…

- We identify the following requirements for *Adaptive Image Offloading under Timing Constraints*.
  - High classification accuracy
  - High encoding efficiency
  - Adaptability to different deadlines and varying bandwidth

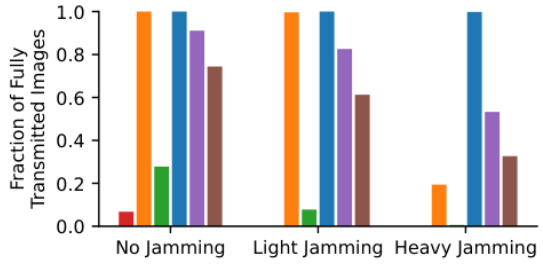
Results (High classification accuracy)

- *Classification* accuracy.



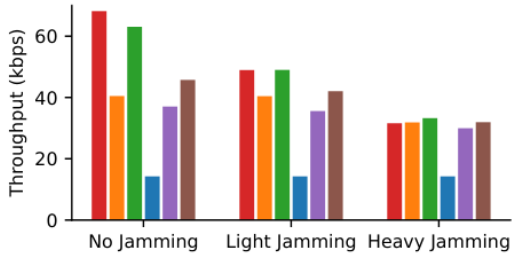
(a) T = 300ms (b) T = 500ms (c) T = 700ms

- Ratio of images fully offloaded.



(b) T = 500ms

- Throughput



(b) T = 500ms

Results (High encoding efficiency)

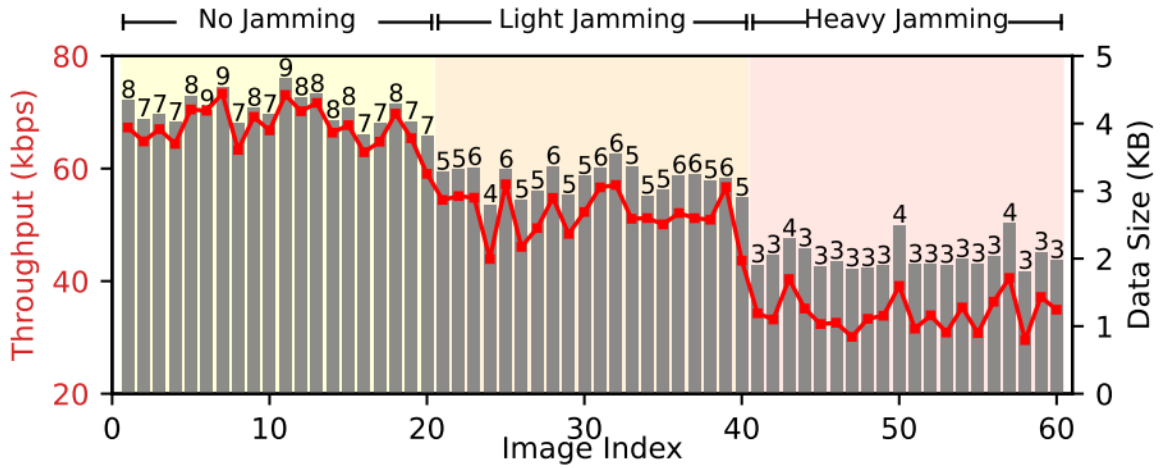
- Encoding efficiency

	Method	Configuration	Average Encoding Overhead (ms)
	PNC	#thread $= 1$	11.8
Non-neural-network-based	WebP	$m = 0, q = 0$	6.0
		$m = 0, q = 20$	6.7
		$m = 6, q = 0$	32.5
		$m = 6, q = 20$	48.6
	Prog. JPEG	$q = 30$	5.8
Neural-network-based	Starfish	One Patch	62.9
	RNN-TFLite	One Iteration	1900

- m represents the *method* parameter in WebP.
- q represents the quality factor.

Results (Adaptability deadlines and bandwidths)

- Adaptation to different deadlines and varying bandwidth



A typical run of the offloading process under varying network bandwidth.

- Red curve: throughput of offloading traffic.
- Gray bars: data size of the fully offloaded features.
- Numbers: the number of features fully offloaded to the edge server.

Results (Adaptability deadlines and bandwidths)

- Adaptation to different deadlines and varying bandwidth

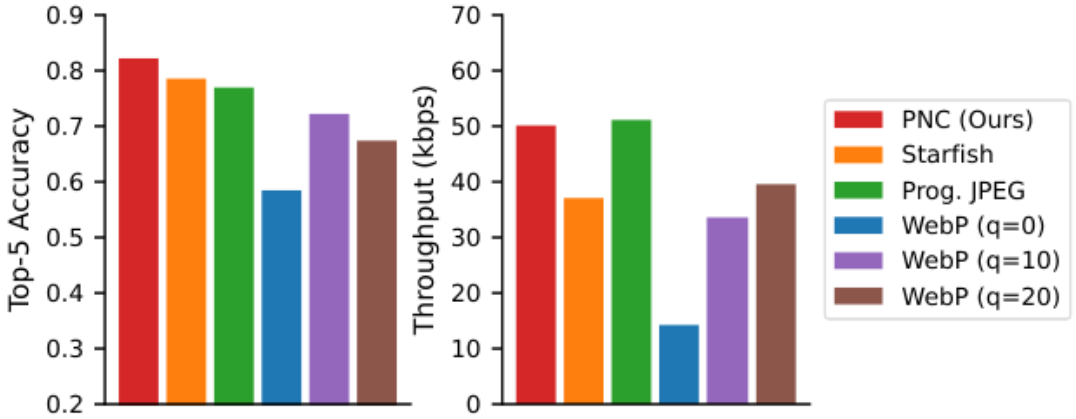


Image classification accuracy and offloading throughput under varying scenarios.

Conclusion

Conclusion

- The Progressive Neural Compression (PNC) framework is tailored for **adaptive image offloading** under timing constraints on embedded IoT systems.
- The **Taildrop** technique helps the system make accurate classification given any data received.
  - Graceful performance degradation in the face of decreasing bandwidth or shorter deadline.
- Experiment shows PNC's outstanding properties for *Adaptive Image Offloading under Timing Constraints*.
  - High classification accuracy
  - High encoding efficiency
  - Adaptability to different deadlines and varying bandwidth